Non-interactive publicly verifiable distributed key generation and resharing

Path to a fully distributed, secure, yet regulated financial word

What will be talk about?

A specific algorithm, that enables the generation of a shared public key in a distributed way, without a central actor, so that a given number of private keys – secretly generated by the participants in the generation process – can be used to create a digital signature that can be authenticated and verified with the shared public key.

Table of contents

NPVDKG-RS

PART I.

Publicly Verifiable Secret Sharing

Definitions

Let's see what's in the rabbit hole
Refresh our memory

- $\mathbb{Z}p$: a finite field, p is a prime number.
  The filed elements are [0, 1, 2, 3 … (p-1)], (y = x mod p)
- (n, t): t-of-n or (n, t) threshold secret sharing enables to a dealer creating $s_1, s_2 \dots s_n$ shares from a secret s, such that any t shares are enough to compute the original secret s, while $t - 1$ shares do not reveal any information about the secret s.
- G_1: group over a field (BLS12-381) and g_1: generator of G_1
- G_2: group over a field (BLS12-381) and g_2: generator of G_2
- e: a non-degenerate, efficiently computable, bilinear pairing function between G_1 and G_2, e: $G_1 \times G_2 \Rightarrow G_t$; other words: $e(g_1, g_2)$ generates G_t.
- G: a point on the elliptic curve over G_1
- Q: a point on the elliptic curve over G_2
- Shamir Secret Sharing (Lagrange evaluation and interpolation)

Lagrange polynomial and interpolation
Step 1 – Define $f(x)$

- Define a $(t-1)$ degree random polynomial over $\mathbb{Z}p$:

$$f(x) = a_{t-1} * x^{t-1} + a_{t-2} * x^{t-2} + \cdots + a_1 * x^1 + a_0, mod\ p$$

$$f(x) = \sum_{k=0}^{t-1} a_k * x^k, mod\ p$$

- Any a_k can be calculated with the Lagrange interpolation polynomials $a(x)$ over $\mathbb{Z}p$.

Lagrange polynomial and interpolation
Step 2 – Define coefficients from points of $f(x)$

- Given t points:

$$(x_0, y_0), \dots, (x_i, y_j), \dots, (x_{t-1}, y_{t-1}); \;\; \forall\, y_k = f(x_k), 0 \le k < t$$

- Then $a(x)$ function can be written as:

$$a(x) = a_x = \sum_{j=0}^{t-1} y_j * l_j(x)\,, mod\ p$$

$$l_j(x) \;=\; \prod_{\substack{0 \le i \le t-1 \\ i \ne j}} \frac{x - x_i}{x_j - x_i}, mod\ p$$

Lagrange polynomial and interpolation
Step 3 – Calculate a_0

- The easiest way to calculate a_0 if we calculate the value of $f(x)$ at $x = 0$. If we given t points, then a_0 is:

$$(x_0, y_0), \dots, (x_i, y_j), \dots, (x_{t-1}, y_{t-1}); \ \forall\, y_k = f(x_k), 0 \le k < t$$

$$a(0) \ = \sum_{j=0}^{t-1} y_j * l_j(0)\,, mod\ p$$

$$l_j(0) \ = \prod_{\substack{0 \le i \le t-1 \\ i \ne j}} \frac{0 - x_i}{x_j - x_i} = \prod_{\substack{0 \le i \le t-1 \\ i \ne j}} \frac{x_i}{x_i - x_j}, mod\ p$$

Lagrange polynomial and interpolation Step 3 – Calculate a_0 final form

$$(x_0, y_0), \dots, (x_i, y_j), \dots, (x_{t-1}, y_{t-1});\ \ \forall\, y_k = f(x_k), 0 \leq k < t$$

$$a_0 = a(0) = \sum_{j=0}^{t-1} y_j * \prod_{\substack{0 \leq i \leq t-1 \\ i \neq j}} \frac{x_i}{x_i - x_j}, mod\ p$$

Shamir Secret Sharing
Share method

- To share a secret s use the previously defined $f(x)$ function define the share function:

  $\boldsymbol{Share(n, t, s, [id_0, id_{t-1}])} \Rightarrow (\boldsymbol{sh_{id_0}}, \dots, \boldsymbol{sh_{id_{n-1}}})$: $\forall\, id_i \neq 0, 0 \leq i \leq t - 1$, set $a_0 = s, s \in \mathbb{F}p$, pick randomly $a_1, \dots, a_{t-1}$ from $\mathbb{Z}p$ and define $f(x) = \sum_{k=0}^{t-1} a_k x^k \,, mod\ p$.
  Return $\left(sh_{id_0}, \dots, sh_{id_{n-1}}\right) = (f(id_0), \dots, f(id_{n-1}))$. (N number of point)

Shamir Secret Sharing
Recover method

- To recover secret s just need to calculate a_0 with the previously shown formula minimum t number of share (sh):
- $\boldsymbol{Recover}\left(\left[\boldsymbol{sh_{id_0}}, \ldots, \boldsymbol{sh_{id_{t-1}}}\right], \left[\boldsymbol{id_0}, \boldsymbol{id_{t-1}}\right]\right) \Rightarrow \boldsymbol{s}: \forall\, id_i \neq 0, 0 \leq i \leq t-1.$ (In other words, given t number of (x, y) point of the original function)

$$s = a(0) \ = \sum_{j=0}^{t-1} sh_{id_j} * \prod_{\substack{0 \leq i \leq t-1 \\ i \neq j}} \frac{id_i}{id_i - id_j}, mod\ p$$

Return s

Verifiable Secret Sharing
Problem to solve

- Problem of the receiver: did she get a correct share?
- Dealer may sand bad share that does not correspond to the dealing or give so many fake shares to different receiver that they could not recover the real secret.

Verifiable Secret Sharing
Feldman's solution

- Feldman proposed a verifiable secret share (VSS) to deal with this problem. His proposed solution uses a $\mathbb{G}$ or order p. The receivers distributes shares together with public group elements $A_0 = g^{a_0}, \dots, A_{t-1} = g^{a_{t-1}}$. (Remember: $a_0 = s, s \in \mathbb{Z}p$, $a_1, \dots, a_{t-1}$ are random from $\mathbb{Z}p$ and are kept in secret.)
- Now "id_i" receiver may check $sh_{id_i} = f(id_i)$ since the correct share satisfy

$$g^{sh_{id_i}} = g^{f(id_i)} = g^{\sum_{k=0}^{t-1} a_k * (id_i)^k} = \prod_{k=0}^{t-1} g^{a_k (id_i)^k} =$$

$$= \prod_{k=0}^{t-1} {A_k}^{(id_i)^k} = A_0 * {A_1}^{(id_i)^1} * {A_2}^{(id_i)^2} * \cdots * {A_{t-1}}^{(id_i)^{t-1}}, mod\ p$$

Verifiable Secret Sharing
Can be publicly verifiable?

- Many verifiable secret sharing protocol use Feldman's related idea. Usually, these protocols let the receiver issue a complaint in case his share is wrong (not satisfy the check). It means, that these protocols have more than one communication rounds, they are interactive.
- Instead of digging deep and create an interactive protocol, we construct a publicly verifiable secret sharing (PVSS) scheme where it is immediately verifiable to everybody, not just the receiver, whether a share is correct or not. Furthermore, it is a non-interactive solution.

Publicly Verifiable Secret Sharing
By definition

- A secret sharing mechanism is publicly verifiable if it is a verifiable secret sharing scheme and if any party (not just the participants of the protocol) can verify the validity of the shares distributed by the dealer
- the object is to resist malicious players, such as:
  I. a dealer sending incorrect shares to some or all of the participants, and
  II. participants submitting incorrect shares during the reconstruction protocol, cf. [CGMA85].

Publicly Verifiable Secret Sharing
In general – Distribution

Distribution of secret s shares is performed by the dealer D, which does the following:

- The dealer creates $sh_{id_0}, sh_{id_1}, \ldots, sh_{id_{n-1}}$ for each participant $P_{id_0}, P_{id_1}, \ldots, P_{id_{n-1}}$ respectively.
- The dealer publishes:
  - the encrypted share $E_{id_i}(sh_{id_i})$ for each P_{id_i}.
  - public group elements $A_0 = g^{a_0}, \ldots, A_{t-1} = g^{a_{t-1}}$. (Remember: $a_0 = s, s \in \mathbb{Z}p$, $a_1, \ldots, a_{t-1}$ are random from $\mathbb{Z}p$ and are kept in secret.) (Feldman's solution)
  - $Proof_{id_0}$

Publicly Verifiable Secret Sharing
In general – Verification

- Anybody knowing the public keys for the encryption methods E_{id_i}, can verify the shares.
- The $Proof_{id_i}, 0 \leq i < n$:
  - shows that each $E_{id_i}(sh_{id_i})$ encrypts sh_{id_i}, $0 \leq i < n$ using Feldman's idea.
  - guarantees that the encrypted share can be decrypted by the receiver participant
  - the reconstruction protocol will result in the same secret s.
- If one or more verifications fails, the protocol is aborted.

Publicly Verifiable Secret Sharing In general – Reconstruction

- After getting all the published data each participant:
  - Executes the verifications process
  - If the verification process fails than stop
  - P_{id_i} decrypts their share of the secret sh_{id_i} using its own private key and $E_{id_i}(sh_{id_i})$.
- t number of n participant P_{id_i} with their sh_{id_i} the secret s can be reconstructed with Shamir secret sharing recovery function.

Our Publicly Verifiable Secret sHaring - PVSH

- Our PVSS called PVSH and uses the ideas from the previous slides:
- Shamir Secret Sharing
- Feldman’s solution
- Publicly verifiable secret sharing in general.

First define our encrypt, verify and decrypt method then we formalize our scheme

PVSH Encrypt, Verify, Decrypt Notations

- id: the receiver participant's public identifier
- sk: the receiver's secret key (PCSRNG), (Fr)
- PK: the receiver's public key, $PK = {g_2}^{sk}$
- sh: secret share, the plain text
- PH: public key of sh, $PH = {g_2}^{sh}$
- $e(A, B) \rightarrow G_t$, "A" is a point at G_1, "B" is a point at G_2 and "e" is a pairing function as described earlier.
- $Hash$: is a hash function
- $HashToG1$: a function, which hashes the input parameter and maps a point on G_1

PVSH Encrypt
Input parameters: id, PK, sh

1. Let $r = random$ (CSPRNG)
2. Let $Q = HashToG1(id, PK)$
3. Let $eh = Hash\big(e(Q,\ PK^r)\big)$
4. Let $c = sh + eh$ (cipher text)
5. Let $U = {g_2}^r$ (public part of r) (used to decode)
6. Let $H = HashToG1(Q, c, U)$
7. Let $V = H^{eh/r}$ (digital signature of H with eh/r) (use to verify)
8. **Return** $(\boldsymbol{c}, \boldsymbol{U}, \boldsymbol{V})$

PVSH Verify
Input parameters: $id, PK, PH, (c, U, V\ )$

1. Let $Q = HashToG1(id, PK)$
2. Let $H = HashToG1(Q, c, U)$
3. Let $e_1 = e(H, {g_2}^c)$ and let $e_2 = e(H, PH) \times e(V, U)$
4. **IF $e1 \neq e2$ RETURN Error Else RETURN OK**

True means:

1. The owner of sk must be able to decode the cipher text with U
2. The cipher text c must contain the secret part sh of PH

False means: (c, U, V) are invalid to each other

PVSH Verify
Proof of correctness, given: $id, PK, PH, (c, U, V\ )$

1. Let $Q = HashToG1(id, PK)$
2. Let $H = HashToG1(Q, c, U)$
3. $e(H, {g_2}^c)$, use identities
   $\Rightarrow e(H, g_2)^c$, use $c = sh + eh$
   $\Rightarrow e(H, g_2)^{sh+eh} \Rightarrow e(H, g_2)^{sh} \times e(H, g_2)^{\frac{eh}{r}r}$ use identities
   $\Rightarrow e\left(H, {g_2}^{sh}\right) \times e\left(H^{eh/r}, {g_2}^r\right)$, use $V = H^{eh/r}, U = {g_2}^r, PH = g2^{sh}$
   $\Rightarrow e(H, PH) \times e(V, U)$

PVSH Decrypt
Input parameters: $id, PK, sk, (c, U, V)$

1. Let $Q = HashToG1(id, PK)$
2. Let $eh' = Hash\left(e\left(Q^{sk}, U\right)\right)$
3. Let $sh' = c - eh'$
4. **Return sh'**

PVSH Decrypt
Proof of correctness, given: $id, PK, sk, (c, U, V)$

Statement: $sh' = sh$

Let $Q = HashToG1(id, PK)$

1. $eh = Hash\left(e(Q,\ PK^r)\right)$, use $PK = {g_2}^{sk}$

   $\Rightarrow Hash\left(e\left(Q,\ {g_2}^{sk*r}\right)\right)$, use identities

   $\Rightarrow Hash\left(e\left(Q^{sk}, {g_2}^{r}\right)\right)$, use $U = {g_2}^r$

   $\Rightarrow Hash\left(e\left(Q^{sk}, U\right)\right) = eh'$

2. $\Rightarrow c = sh + eh \Rightarrow c = sh + eh' \Rightarrow sh = c - eh' \Rightarrow sh = sh'$

Some observation

- To decode you need to know sk, r or eh. To calculate r or eh from U or V, $U = {g_2}^r, V = H^{eh/r}$ is hard since this is a discrete logarithm problem. The private key sk is owned by the recipient and kept in secret.
- To decode you need to know somehow $e(Q, g_2)^{sk*r}$
- $z_1 = PK * U = {g_2}^{sk} * {g_2}^r = {g_2}^{sk+r}$, you can't get $sk * r$, dlog problem.
- $z_2 = e(V, U) = e(H, g_2)^{\frac{eh}{r}*r} = e(H, g_2)^{eh}$, you can't get eh, dlog problem.
- **Since $\boldsymbol{Q = HashToG1(id, PK)}$, so NEVER digitally sign $\boldsymbol{Q}$ with $\boldsymbol{sk}$.** Because knowing Q^{sk} and with the give $U = {g_2}^r$ leads to easy decoding!

Proof V is a zero-knowledge proof?

Zero knowledge definition: **if the statement is true, no verifier learns anything other than the fact that the statement is true.** In other words, just knowing the statement (not the secret) is sufficient to imagine a scenario showing that the prover knows the secret.

The PVSHencrypt method result verification vector V does not add more information to the verifier but V can be used to verify that the statement is true. **Statement: with** (c, U, V) **the receiver** (owner of PK knows sk) **must be able to decode cipher text and the cipher text contains the secret part of** PH**.**

End of PART I.

NPVDKG-RS

PART II.

Publicly Verifiable Secret Sharing

Methods summary

PVSH – the full algorithm
Share method

$\boldsymbol{Share(n, t, [a_0, \ldots, a_{t-1}], [id_0, \ldots, id_{n-1}])} \Rightarrow$
$(\left[\boldsymbol{sh_{id_0}, \ldots, sh_{id_{n-1}}}\right], [\boldsymbol{A_0, \ldots, A_{t-1}}])$: $\forall\, id_i \neq 0, 0 \leq i \leq n-1, (a_0 = s, s \in \mathbb{Z}p$ is a secret to share and $a_1, \ldots, a_{t-1}$ are randomly picked from $\mathbb{Z}p$)
Define $f(x) = \sum_{k=0}^{t-1} a_k x^k\, , mod\ p$.
Return:

$$\left[sh_{id_0}, \ldots, sh_{id_{n-1}}\right] = [f(id_0), \ldots, f(id_{n-1}))]$$
$$[A_0, \ldots, A_{t-1}] \ = [g^{a_0}, \ldots, g^{a_{t-1}}]$$

(Note: previously we defined A_i, later we call it PG_i. Also keep in mind that a_i are sg_i. Never mind, later we will define it.)

PVSH – the full algorithm PVSHEncrypt method

$\boldsymbol{PVSHEncrypt}(\boldsymbol{id_i}, \boldsymbol{PK_{id_i}}, \boldsymbol{sh_{id_i}}) \Rightarrow (\boldsymbol{c_{id_i}}, \boldsymbol{U_{id_i}}, \boldsymbol{V_{id_i}}) : \forall\, id_i \neq 0, 0 \leq i \leq n-1.$
The id_i is the a public identifier of the i -th receiver participant and PK_i is the public key of the i -th receiver participant.

$$r = random, Q = HashToG1(id_i), eh = Hash\left(e\left(Q, {PK_{id_i}}^r\right)\right)$$

Return:

$$c_{id_i} = sh_{id_i} + eh$$
$$U_{id_i} = {g_2}^r$$
$$V_{id_i} = \left(HashToG1(Q, c, U)\right)^{eh/r}$$

PVSH – the full algorithm PVSHVerify method

$\boldsymbol{PVSHVerify}\left(\boldsymbol{id_i}, \boldsymbol{PK_{id_i}}, \boldsymbol{PH_{id_i}}, \left(\boldsymbol{c_{id_i}}, \boldsymbol{U_{id_i}}, \boldsymbol{V_{id_i}}\right)\right) \Rightarrow \vdash \mid \dashv : \forall\, id_i \neq 0, 0 \leq i \leq n-1$. $\vdash$ means OK, $\dashv$ means Error. The id_i is the public identifier of the i -th receiver participant and $\left(c_{id_i}, U_{id_i}, V_{id_i}\right)$ is the encrypted share of the i -th receiver participant.

$$Q = HashToG1\left(id_i, PK_{id_i}\right), H = HashToG1(Q, c, U)$$

$$e_1 = e(H, {g_2}^c), e_2 = e\left(H, PH_{id_i}\right) \times e\left(V_{id_i}, U_{id_i}\right)$$

Return:

$$if\ e1 \neq e2\ RETURN \dashv ELSE\ RETURN \vdash$$

PVSH – the full algorithm PVSHDecrypt method

$\boldsymbol{PVSHDecrypt\left(id_i, PK_{id_i}, sk_{id_i}, \left(c_{id_i}, U_{id_i}, V_{id_i}\right)\right) \Rightarrow sh_{id_i}}: \forall\, id_i \neq 0, 0 \leq i \leq n-1$. The id_i is the public identifier, the PK_{id_i} is the public key and sk_{id_i} is the private key and $\left(c_{id_i}, U_{id_i}, V_{id_i}\right)$ is the encrypted share of the i -th receiver participant.

$$Q = HashToG1\left(id_i, PK_{id_i}\right)$$

$$eh' = Hash\left(e\left(Q^{sk_{id_i}}, U\right)\right)$$

Return:

$$sh_{id_i} = c_{id_i} - eh'$$

PVSH – the full algorithm
Recover share method

$\boldsymbol{Recover}\left(\left[\boldsymbol{sh_{id_0}}, \dots, \boldsymbol{sh_{id_{t-1}}}\right], \left[\boldsymbol{id_0}, \dots, \boldsymbol{id_{t-1}}\right]\right) \Rightarrow \boldsymbol{s}\colon \forall\, id_i \neq 0, 0 \leq i \leq t-1$. (In other words, given t number of (x, y) point of the original function)

$$s = a(0) \; = \sum_{j=0}^{t-1} sh_{id_j} * \prod_{\substack{0 \leq i \leq t-1 \\ i \neq j}} \frac{id_i}{id_i - id_j}, mod\ p$$

Return s

End of PART II.

NPVDKG-RS

PART III.

Non-Interactive Publicly Verifiable Distributed key generation and resharing

The full algorithm

Algorithm - from a bird's eye view

A goal is a t-of-n or (n, t) Threshold distributed key generation, which is non-interactive and publicly verifiable

I. Setup phase (privately)
II. Publication of the result of the Setup phase
III. Key generation (privately) from the publicly available data

Preliminary

The followings are given:

- N number of participants want to create a t-of-n or (n, t) threshold signature public key and privately create a valid secret key of itself.
- Every participant knows all id_i and PK_{id_i} of the participants
- Each participant have its own id_i, sk_{id_i} and PK_{id_i}
- $1 \leq t \leq n$, note: if $t = 1$ means that all participants will end up the same secret key, it can be useful in some application.

I. Setup phase – Creating the "gifts" Each participant privately, i-th participant

1. Generate a random number s_i (in case of resharing s_i is the old sh secret key) and call $\boldsymbol{Share}(\boldsymbol{n}, \boldsymbol{t}, [\boldsymbol{s_i}, \boldsymbol{a_1}, \dots, \boldsymbol{a_{t-1}}], [\boldsymbol{id_0}, \dots, \boldsymbol{id_{n-1}}]) \Rightarrow \left(\left[\boldsymbol{sh_{i,0}}, \dots, \boldsymbol{sh_{i,j}}, \dots, \boldsymbol{sh_{i,n-1}}\right], \left[\boldsymbol{PG_{i,0}}, \dots, \boldsymbol{PG_{i,k}}, \dots, \boldsymbol{PG_{i,t-1}}\right]\right)$: $\forall\, id_j \neq 0, \forall i, j \in [0, n-1], \forall k \in [0, t-1]$. The $a_1, \dots, a_{t-1}$ are randomly picked. The i-th participant create shares to all participant (including itself)
   Note: $\left[PG_{i,0}, \dots, PG_{i,t-1}\right]$ original notation is $[A_0, \dots, A_{t-1}]$
2. To $\forall\, \boldsymbol{sh_{i,j}}$ call $\boldsymbol{PVSHEncrypt}\left(\boldsymbol{id_j}, \boldsymbol{PK_{id_j}}\, \boldsymbol{sh_{i,j}}\right) \Rightarrow \left(\boldsymbol{c_{id_{i,j}}}, \boldsymbol{U_{id_{i,j}}}, \boldsymbol{V_{id_{i,j}}}\right)$: $\forall i, j \in [0, n-1]$).
   Note: The i-th participant generated and encrypted a share to j-th participant.

II. Publication – Place all the gifts under the Christmas tree, *i*-th participant

1. All participant publish, $\forall i,j \in [0, n-1]), \forall k \ \in [0, t-1]$:

$$\left[\boldsymbol{PG_{i,0}}, \ldots, \boldsymbol{PG_{i,k}}, \ldots, \boldsymbol{PG_{i,t-1}}\right], \left(\boldsymbol{c_{id_{i,j}}}, \boldsymbol{U_{id_{i,j}}}, \boldsymbol{V_{id_{i,j}}}\right)$$

Note: in some slides for a shorter writing, we write $ESH_{i,j}$, where

$ESH_{i,j} = \left(c_{id_{i,j}}, U_{id_{i,j}}, V_{id_{i,j}}\right)$

Everybody knows which participant created the data (sender) and contains the recipient, too.

III. Key generation – gift breakdown – 1 Each participant privately, i-th participant

1. Get all data: $\left[PG_{i,0}, \dots, PG_{i,k}, \dots, PG_{i,t-1}\right], \left(c_{id_{i,j}}, U_{id_{i,j}}, V_{id_{i,j}}\right)$ and known to public id_i, PK_i and all id_i Participant knows her secret key sk_i. $\forall i, j \in [0, n-1]), \forall k \in [0, t-1]$
2. Calculate $\boldsymbol{PH_{i,j}}$: $\boldsymbol{Share}\left(\boldsymbol{n}, \boldsymbol{t}, \left[\boldsymbol{PG_{i,0}}, \dots, \boldsymbol{PG_{i,k}}, \dots, \boldsymbol{PG_{i,t-1}}\right], \left[\boldsymbol{id_0}, \dots, \boldsymbol{id_j}, \dots, \boldsymbol{id_{n-1}}\right]\right) \Rightarrow \left(\left[\boldsymbol{PH_{i,0}}, \dots, \boldsymbol{PH_{i,j}}, \dots, \boldsymbol{PH_{i,n-1}}\right], [\dots \boldsymbol{not\ interested} \dots]\right)$, $\forall i, j \in [0, n-1]), \forall k \in [0, t-1]$
3. All participant: $\forall i, j$ call $\boldsymbol{PVSHVerify}\left(\boldsymbol{id_i}, \boldsymbol{PK_i}, \boldsymbol{PH_{i,j}}, \left(\boldsymbol{c_{id_{i,j}}}, \boldsymbol{U_{id_{i,j}}}, \boldsymbol{V_{id_{i,j}}}\right)\right) \Rightarrow \vdash \mid \dashv$

   If any verification process returns $\dashv$, then the j is the sender who tries to trick the recipient i. Abort the process. Because the data are public, easily can everybody check this.

III. Key generation – gift breakdown – 2 Each participant privately, *i*-th participant

4. The i-th participant: $\forall\, j$ call $\boldsymbol{PVSHDecrypt}\left(\boldsymbol{id_i}, \boldsymbol{PK_i}, \boldsymbol{sk_i}, \left(\boldsymbol{c_{id_{i,j}}}, \boldsymbol{U_{id_{i,j}}}, \boldsymbol{V_{id_{i,j}}}\right)\right) \Rightarrow \boldsymbol{sh_{ij}}$
5. The i-th participant to recover her secret key call $\boldsymbol{Recover}([\boldsymbol{sh_{i,0}}, \dots, \boldsymbol{sh_{i,j}}, \dots, \boldsymbol{sh_{i,n-1}}], [\boldsymbol{id_0}, \dots, \boldsymbol{id_{n-1}}]) \Rightarrow \boldsymbol{sh_i}, \forall j \in [0, n-1])$
6. The i-th participant to calculate all public keys call $\forall i, j \in [0, n-1])$: $\boldsymbol{Recover}([\boldsymbol{PH_{i,0}}, \dots, \boldsymbol{PH_{i,j}}, \dots, \boldsymbol{PH_{i,n-1}}], [\boldsymbol{id_0}, \dots, \boldsymbol{id_{n-1}}]) \Rightarrow \boldsymbol{PH_i}$, Note: PH_i can be calculated from sh_i if i contains itself.
7. The i-th participant call $\boldsymbol{Recover}([\boldsymbol{PH_i}, \dots, \boldsymbol{PH_{n-1}}], [\boldsymbol{id_0}, \dots, \boldsymbol{id_{n-1}}]) \Rightarrow \boldsymbol{PG}, \forall i \in [0, n-1])$, Note: PG will be the same to all participant

End of PART III.

NPVDKG-RS

PART IV.

Non-Interactive Publicly Verifiable Distributed key generation and resharing

Graphical representation of steps

I. Setup phase – Step 1
Choose random numbers

$$[SG_{i,0}, SG_{i,1}, \ldots, SG_{i,j}] = [s_i, a_1, \ldots, a_{t-1}]$$

Private

Participant 1	SG1	SH1	PG1	ESH1
ID1 **SM1 - PM1**	SG11 SG12			

Participant 2	SG2	SH2	PG2	ESH2
ID2 **SM2 - PM2**	SG21 …			

Participant 3	SG3	SH3	PG3	ESH3
ID3 **SM3 - PM3**				

Public

I. Setup phase – Step 1
Generate secret shares (Lagrange evaluation)

Private

Participant 1	SG1	SH1	PG1	ESH1
ID1 **SM1 - PM1**	SG11 SG12	SH11		
		SH12		
		SH13		

Participant 2	SG2	SH2	PG2	ESH2
ID2 **SM2 - PM2**	SG21 SG22	SH21		
		SH22		

Participant 3	SG3	SH3	PG3	ESH3
ID3 **SM3 - PM3**	SG31 SG32			

Public

I. Setup phase – Step 1
Share calculate $\left[PG_{i,0}, \ldots, PG_{i,j}\right]$

Private

Participant 1	SG1	SH1	PG1	ESH1
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	
		SH12		
		SH13		

Participant 2	SG2	SH2	PG2	ESH2
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 …	
		SH22		
		SH23		

Participant 3	SG3	SH3	PG3	ESH3
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 …	
		SH32		
		SH33		

Public

I. Setup phase – Step 2
Encrypt all $sh_{i,j}$

Private

Participant 1	**SG1**	**SH1**	**PG1**	**ESH1**
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	**SG2**	**SH2**	**PG2**	**ESH2**
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		

Participant 3	**SG3**	**SH3**	**PG3**	**ESH3**
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		
		SH33		

Public

II. Every member publish

Private

Participant 1	SG1	SH1	PG1	ESH1
ID1 SM1 - PM1	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	SG2	SH2	PG2	ESH2
ID2 SM2 - PM2	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	SG3	SH3	PG3	ESH3
ID3 SM3 - PM3	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public Shared contributions*

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22

*Every contribution must be signed by the sender!

Setup and publication phase ready

Private

Participant 1	SG1	SH1	PG1	ESH1
ID1 SM1 - PM1	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	SG2	SH2	PG2	ESH2
ID2 SM2 - PM2	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	SG3	SH3	PG3	ESH3
ID3 SM3 - PM3	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

III. Key generation – Step 1
Get all data (Example $i = 2$)

Private

Participant 1	**SG1**	**SH1**	**PG1**	**ESH1**
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	**SG2**	**SH2**	**PG2**	**ESH2**
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	**SG3**	**SH3**	**PG3**	**ESH3**
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

ID1	PM1	ESH11	ESH12		ESH13	PG11	PG12			
ID2	PM2	ESH21	ESH22		ESH23	PG21	PG22			
ID3	PM3	ESH31	ESH32		ESH33	PG31	PG32			

III. Key generation – Step 2
Calculate $PH_{i,j}$ (Lagrange evaluation)

Private

Participant 1	SG1	SH1	PG1	ESH1
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	SG2	SH2	PG2	ESH2
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	SG3	SH3	PG3	ESH3
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

ID1	PM1	ESH11	ESH12		ESH13	PG11	PG12	PH11	PH12	PH13
ID2	PM2	ESH21	ESH22		ESH23	PG21	PG22	PH21	PH22	
ID3	PM3	ESH31	ESH32		ESH33	PG31	PG32			

III. Key generation – Step 3
Verify all $PH_{i,j}, ESH_{i,j}$

Private

Participant 1	**SG1**	**SH1**	**PG1**	**ESH1**
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	**SG2**	**SH2**	**PG2**	**ESH2**
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	**SG3**	**SH3**	**PG3**	**ESH3**
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

ID1	PM1	ESH11	ESH12		ESH13	PG11	PG12	PH11	PH12	PH13
ID2	PM2	ESH21	ESH22		ESH23	PG21	PG22	PH21	PH22	PH23
ID3	PM3	ESH31	ESH32		ESH33	PG31	PG32	PH31	PH32	PH33

III. Key generation – Step 4
Decrypt all $ESH_{2,j}$ (because example $i = 2$)

Private

Participant 1	**SG1**	**SH1**	**PG1**	**ESH1**
ID1 SM1 - PM1	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	**SG2**	**SH2**	**PG2**	**ESH2**
ID2 SM2 - PM2	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	**SG3**	**SH3**	**PG3**	**ESH3**
ID3 SM3 - PM3	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

			ID1	ID2	ID3					
ID1	PM1	ESH11	ESH12	SH12	ESH13	PG11	PG12	PH11	PH12	PH13
ID2	PM2	ESH21	ESH22	SH22	ESH23	PG21	PG22	PH21	PH22	PH23
ID3	PM3	ESH31	ESH32		ESH33	PG31	PG32	PH31	PH32	PH33

III. Key generation – Step 5 – Step 6 – Step 7 Recovery SH_2, PH_j, PG (Lagrange interpolation)

Private

Participant 1	**SG1**	**SH1**	**PG1**	**ESH1**
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	**SG2**	**SH2**	**PG2**	**ESH2**
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	**SG3**	**SH3**	**PG3**	**ESH3**
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

ID1	PM1	ESH11	ESH12	SH12	ESH13	PG11	PG12	PH11	PH12	PH13
ID2	PM2	ESH21	ESH22	SH22	ESH23	PG21	PG22	PH21	PH22	PH23
ID3	PM3	ESH31	ESH32	SH32	ESH33	PG31	PG32	PH31	PH32	PH33
				sh2		PG		ph1		

Participant 2: Key generation ready

Private

Participant 1	**SG1**	**SH1**	**PG1**	**ESH1**
ID1 **SM1 - PM1**	SG11 SG12	SH11	PG11 PG12	ESH11
		SH12		ESH12
		SH13		ESH13

Participant 2	**SG2**	**SH2**	**PG2**	**ESH2**
ID2 **SM2 - PM2**	SG21 SG22	SH21	PG21 PG22	ESH21
		SH22		ESH22
		SH23		ESH23

Participant 3	**SG3**	**SH3**	**PG3**	**ESH3**
ID3 **SM3 - PM3**	SG31 SG32	SH31	PG31 PG32	ESH31
		SH32		ESH32
		SH33		ESH33

Public

ID1	PM1	ESH11	ESH12	ESH13	PG11	PG12
ID2	PM2	ESH21	ESH22	ESH23	PG21	PG22
ID3	PM3	ESH31	ESH32	ESH33	PG31	PG32

ID1	PM1	ESH11	ESH12	SH12	ESH13	PG11	PG12	PH11	PH12	PH13
ID2	PM2	ESH21	ESH22	SH22	ESH23	PG21	PG22	PH21	PH22	PH23
ID3	PM3	ESH31	ESH32	SH32	ESH33	PG31	PG32	PH31	PH32	PH33
				sh2		**PG**		**ph1**	**ph2**	**ph3**

NPVDKG-RS

Thanks for your attentions

Contact: info@natrix.io

https://www.linkedin.com/company/natrix-blockchain-platform/

Thanks to Andras Szabolcsi for creating the algorithm

https://www.linkedin.com/in/andras-szabolcsi/